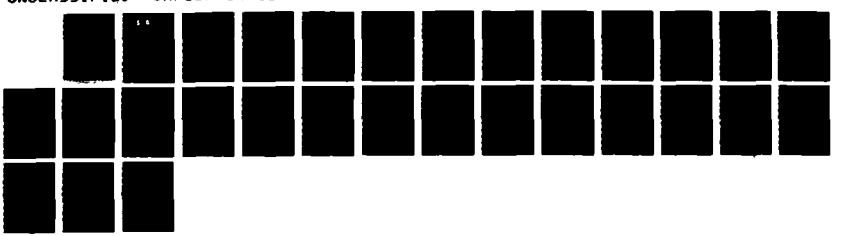
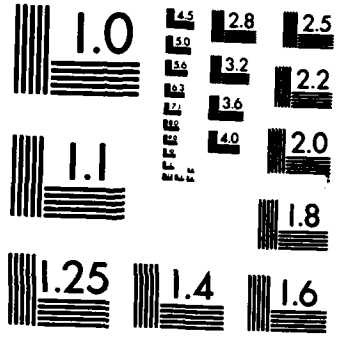
AD-A185 275 COSTS OF QUADTREE REPRESENTATION OF NON-DENSE MATRICES (U) INDIANA UNIV AT BLOOMINGTON DEPT OF COMPUTER SCIENCE D S WISE ET AL AUG 87 AFOSR-TR-87-1168 1/1

UNCLASSIFIED $AFOSR-84-0372 F/G 12/1 NL





Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A185 275

DTIC FILE COPY (2)

REPORT DOCUMENTATION PAGE

DTIC ELECTE S OCT 0 1 1987 D

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-1168
6a. NAME OF PERFORMING ORGANIZATION: Indiana University	6b. OFFICE SYMBOL (If applicable):
6c. ADDRESS (City, State, and ZIP Code): Bloomington, IN 47405	7a. NAME OF MONITORING ORGANIZATION: AFOSR/NM
	7b. ADDRESS (City, State, and ZIP Code): AFOSR/NM Bldg 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR	8b. OFFICE SYMBOL (If applicable): NM
8c. ADDRESS (City, State, and ZIP Code): AFOSR/NM Bldg 410 Bolling AFB DC 20332-6448	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR 84-0372
	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A2; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Costs of Quadtree Representation of Non-dense Matrices

12. PERSONAL AUTHOR(S): David S. Wise and John Franco

13a. TYPE OF REPORT: Technical report	13b. TIME COVERED: FROM 9/30/84 TO 8/87	14. DATE OF REPORT (Year, Month, Day): August, 1987	15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Quadtree representation of matrices offers a homogeneous representation for both sparse and dense matrices, with advantages for processing on multi-processors. This paper offers exact values for the average depth and on the number of nodes in this representation of some familiar patterned matrices: symmetric, triangular, and banded. It similarly measures three permutation matrices as comparative examples of non-dense, unpatterned matrices. Those results are exact values for the shuffle and bit-reversal permutations raised by the fast Fourier transform, as well as tight bounds on the expected values from purely random permutations. Two different measures for density and for sparsity are proposed from these values, and a simple analysis of quadtree matrix addition is given as an illustration of these measures.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL: Maj. John P. Thomas	22b. TELEPHONE (Include Area Code): 767-5025 / 22c. OFFICE SYMBOL: NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

87

Costs of Quadtree Representation of Non-dense Matrices

David S. Wise * & John Franco †

Computer Science Department, Indiana University

101 Lindley Hall, Bloomington, IN 47405-4101

dswise@iuvax.cs.indiana.edu franco@iuvax.cs.indiana.edu

PRELIMINARY VERSION - FIGURES MISSING

CR categories and Subject Descriptors:
E.1 [**Data Structures**]: Trees; G.2.1 [**Combinatorics**]: Recurrences and difference equations; G.1.3 [**Numerical Linear Algebra**]: Sparse and very large systems; F.2.2 [**Nonnumerical Problem Complexity**]: Computations on discrete structures; D.1.1 [**Applicative (Functional) Programming Techniques**].
General Term: Measurement.

Abstract

Quadtree representation of matrices offers a homogeneous representation for both sparse and dense matrices, with advantages for processing on multiprocessors. This paper offers exact values for the *average depth* and on the *number of nodes* in this representation of some familiar patterned matrices: symmetric, triangular, and banded. It similarly measures three permutation matrices as comparative examples of non-dense, unpatterned matrices. Those results are exact values for the shuffle and bit-reversal permutations raised by the fast Fourier transform, as well as tight bounds on the expected values from purely random permutations. Two different measures for *density* and for *sparsity* are proposed from these values, and a simple analysis of quadtree matrix addition is given as an illustration of these measures.

Section 1. Introduction

* Research reported herein was sponsored, in part, by the National Science Foundation under Grant Number DCR 84-05241.

† Research reported herein was sponsored, in part, by the U.S. Air Force under a grant numbered AFOSR 84-0372.

Recent papers [11, 12] have proposed a homogeneous quadtree representation for both dense and sparse matrices, which provides a graceful decomposition of algorithms suitable for scheduling on multiprocessors. Most of the discussion has been directed at the algorithms, themselves, focusing on the importance for decomposing $n \times n$ matrices gracefully onto p processors for $p \ll n$. Some of these algorithms are outlined at the end of the next section.

All the results there apply, as well, to sparse matrix techniques, if we can only show that the quadtree representation is acceptably sparse. That is the goal of this paper. Section 2 defines the quadtree representation of matrices, structuring dense and sparse matrices indistinguishably. This homogeneous representation comes at some price, because above all the scalar entries in a matrix is a large tree of nonterminal nodes, which is supplanted by addressing hardware in the usual, sequential Von Neumann memory.

Although nontrivial in comparison with constant-time access into sequentially stored matrices, the additional overhead from these nonterminal nodes is an artificial concern in a couple of ways. First of all, it may be irrelevant in the algorithms described below, which typically involve recursive descent. That is, rather than accessing elements of a matrix from the root of the tree (analogously to indexing through a conventional array based from a single memory address), these algorithms recurse to nested and successively shallower subtrees, so that an entire path from the root is rarely traversed just to manipulate a *single* element.

Secondly, even if the complete path were traversed upon every probe of an array, the time spent to traverse that path might be recovered in other ways. More specifically, if a heap were spread across very many memory banks and several processors were performing similar algorithms on one globally accessible quadtree-matrix (whose nodes mapped across those banks in a random order), then the aggregate performance of those processors might actually improve over correspondingly similar algorithms on matrices stored sequentially. The improvement arises from the random pattern mitigating the problem of two, or more, processors falling into the same access pattern and addressing the same banks simultaneously and repeatedly. The coincidence of regular access patterns to regularly allocated arrays, even from regular offsets within different matrices, is likely to become an ever increasing problem with more processors. Randomization available from this kind of heap

[4] would not prevent the first contention between two algorithms, but it would certainly help prevent a first from being immediately followed by more. Thus, the tree structure would allow many coprocessors to run with less memory contention, and to absorb the cost of repeated path traversals.

Both efficiency of access and sparse matrices are of high interest in parallel processing. Many caching strategies fail under parallelism, and bus or switch contention compounds the problem of wait states. Many problems sufficiently large and important enough to justify parallel computation also exhibit sparseness. Therefore, these results are of great interest in designing parallel algorithms and parallel computers [2, 8].

It is, therefore, important to measure the costs of large matrices in the quadtree format. The next eight sections give some answers for familiar matrices [9]; many of them are characterized by strong patterning. Section 2 offers the definitions and normal forms for quadtree representation; it also indicates how some parallel algorithms are fitted to this representation. The next section presents the expected path and exact space measures for dense and symmetric matrices. Section 4 treats triangular, "clip," and finally banded matrices, the most familiar of sparse matrices. The "shuffle" and "bit-reversal" permutations, are encountered when studying the fast Fourier transform, are analyzed in Section 5; the latter, here called the *FFT permutation*, is the worst case for these measures. Section 6 gives tight bounds on random permutation matrices, and shows them to behave nearly as badly as the worst case. Based on the foregoing analyses, we propose measures for *sparsity* and *density* in Section 7. As an example of their use, Section 8 offers a worst-case analysis of the performance of matrix addition. The last section offers conclusions.

Section 2. Quadtree Representation and Algorithms

Dimension refers to the number of subscripts on an array. *Order* of a square matrix means the number of its rows or columns when written as the conventional tableau. Similarly, the *size* of a vector is the number of elements when it is expressed in the conventional tuple formulation.

Let any d-dimensional array be represented as a 2^d-ary tree. Here only matrices and vectors are considered, where $d = 2$ suggests quadtrees, and $d = 1$ suggests binary trees.

Matrix algorithms will be arranged so that we may (without loss) perceive any nonzero scalar, x, as a diagonal matrix of arbitrary order, entirely of zeroes except for x's on the main diagonal; that is, $x = [x\delta_{i,j}]$. (This particular compression, however, is ignored in the analyses of this paper.) Thus, a domain is postulated that coalesces scalars and matrices, with every scalar-like object conforming also as a matrix of any order. Of particular interest is the scalar 1, which is at once the *unique* multiplicative identity for scalar/matrix arithmetic. The additive identity, 0, is represented by the null pointer, **NIL** (using PASCAL notation), which is particularly helpful in reducing dereferencing necessary in manipulating sparse matrices.

A matrix (of otherwise-known order) is either a 'scalar' or it is a quadruple of four equally-ordered submatrices. So that this recursive cleaving works smoothly, we embed a matrix of size $n \times n$ in a $2^{\lceil \lg n \rceil} \times 2^{\lceil \lg n \rceil}$ matrix, justified at the lower, right (southeast) corner with zero padding to the north and west. Padding with **NIL** minimizes the space consumed in padding. The matrix is justified to the southeast, rather than the northwest, so to help with computation of eliminants [1].

This prescribes a *normal form* for quadtrees: no scalar entry is ever 0, four quadrants cannot all be **NIL**, and if the southwest and northeast are **NIL** then the northwest and southeast cannot be the same scalar. Similarly, **NIL** as a vector refers to the zero vector, and any non-zero scalar x is interpreted as a vector of arbitrary size, each of whose elements is x; this normal form for vectors precludes any entry from being 0 and any brothers from both being **NIL** or the same scalar.

Inferring the conventional meaning from such a matrix now requires additional information (*viz.* its order), but we can proceed quite far without size information; it only becomes critical upon Input or Output. One must acknowledge that the I/O conversions are non-trivial algorithms [12], but because they consume little processor resource—and are restrained, also, by communication bandwidth—we eschew them here. Like floating-point number conversions, they are an irritating impediment to one who would experiment with the algorithms discussed below.

A "header" above each matrix quadtree should contain its size, necessary for output translation and needed for better control of certain algorithms, like pivoting on singular

matrices. Often, however, its value is not essential to binary operators, especially if conformability checks are unnecessary at run time. Here, size is the proper length of the main diagonal—exclusive of any padding and normalization.

An *optional* annotation is to include a bit within each pointer, indicating that the referenced tree structure is to be interpreted as transposed, recursively interchanging southwest\northeast quadrants upon any access. Call this the *transposed* flag. With it, quadtree representation allow transposition of an entire matrix in constant time—at the cost of building a new reference with that flag inverted. It also indicates how row and column traversal use the same algorithm: a symmetric-order traversal of the appropriately projected binary tree. However, unless hardware support is available, testing of this flag slows dereferencing of every quadrant. Because of this cost, is used in this paper only in comparing measures for the explicit class of symmetric matrices.

Tree-structured memory is well suited to functional or applicative programming style, whose only control structure is function invocation and whose only iteration is expressed through recursion. The recursive definition of a quaternary trees mimics the recursive structure of programs that manipulate them.

Thus, the algorithm for matrix addition [11] decomposes naturally into four quadrant additions which are separate and independent processes. Because of their mutual independence, these four are naturally computed in parallel within a shared memory, or distributed to independent processors with private memory. In the latter case, the tree structure of the matrix may be better mapped onto a tree of private memories. And the division extends naturally to 16, 64, 256, *etc.* processors, or—by splitting the sums in half, rather than in quarters—to 2, 8, 128 *etc.* as well. When either addend is **NIL**, addition returns a shared reference to the other addend.

The problem of matrix multiplication may be decomposed two ways (again treating the product as two halves), four ways (the four quadrants of the answer), and eight ways (the eight quadrant products in Strassen's decomposition [10] of Gaussian matrix multiplication.) Whenever a multiplier or multiplicand is **NIL**, the product is annihilated to **NIL**; if either is 1 the product becomes a reference to the other.

Solutions to linear systems and matrix inversion have been reduced to the Pivot Step algorithm [5], where the "independent" problem of a stable choice for the pivot element

folds naturally onto the tree. The quadtree not only suggests efficient tree-search, but also it provides the structure upon which to bind decorations (local maxima in a sparse matrix) that do not change across many pivot steps, thereby truncating the full search for successive pivots.

The remainder of this paper addresses static measures of quadtree representation, only briefly considering any of the algorithms mentioned above. Some of the efficiencies provided by normal form, moreover, are ignored, because this paper only considers efficiencies due to the occurrences of **NIL** in sparse matrices. It does not analyze, for instance, space efficiencies due to the representation of a 16×16 identity submatrix simply as the scalar 1. Moreover, even though implicitly shared references are likely to result from many programs, none of the analyses that follow consider any sharing beyond that explicitly stated and indicated in the figures. All these measures are, therefore, slightly conservative.

Section 3. Dense and Symmetric Matrices

A matrix is *dense* if it contains no zero entries. It is *symmetric* either if it is a non-zero scalar, or if its northwest and southeast quadrants are symmetric *and* its northeast and southwest pointers are transposes of one-another; they share a reference in the tree representation, except for inverting the "transposed" flag on one. These shared quadrants are presumed to be dense in the analysis that follows. Both these decompositions are illustrated in Figure 1.

Let $n = 2^p$, for p an integer, and define the functions S_D, mapping p to the number of nodes (*space*) necessary to represent a dense $n \times n$ matrix, and P_D, mapping p to the average *path* length in a dense $n \times n$ matrix. Similarly, S_S maps p to the space necessary to represent a symmetric $n \times n$ matrix, and P_S maps p to the average *path* length in a symmetric $n \times n$ matrix.

Then

$$S_D(0) = 1;$$

$$S_S(0) = 1;$$

$$S_D(p+1) = 1 + 4S_D(p);$$

$$S_S(p+1) = 1 + 2S_S(p) + S_D(p).$$

In the last case, the southwest and northeast quadrants are dense and coincident, and, together, they only contribute to the space cost but once. Figure 1 shows this sharing, which requires the use of the optional /it transposed flag. Thence,

$$\begin{aligned} S_D(p) &= \sum_{i=0}^{p} 4^i = \frac{4^{p+1}-1}{3} \\ &= \frac{4n^2-1}{3} = \tfrac{4}{3}(n+\tfrac{1}{2})(n-\tfrac{1}{2}). \\ S_S(p) &= \sum_{i=0}^{p} 2^i - \frac{1}{3}\sum_{i=0}^{p-1} 2^i + \frac{4^p}{3}\sum_{i=0}^{p-1}\frac{2^i}{4^i} = \tfrac{2}{3}(4^p-1)+2^p \\ &= \tfrac{2}{3}(n^2-1)+n = \tfrac{2}{3}(n+2)(n-\tfrac{1}{2}). \end{aligned}$$

In traditional representations using sequentiality of memory addresses, we expect these space requirements to be n^2 and $(n^2+n)/2$, respectively, although additional information is needed for a program to attain the second.

Assuming (simplistically) that scalars and nonterminal nodes all take the same space, we immediately observe a 33% space increase over sequential storage of matrices. In fact, it is likely that the nonterminal nodes (of four pointers) are larger than terminal nodes (a single scalar), indicating an additional consideration—the constant of proportionality—to be considered when comparing *different* representations. This space difference is probably closer to 66% for that reason.

The expense to access an individual element is a less critical measure, because most algorithms do not perform that operation as a primitive fetch from the root of the tree. Rather, the problem decomposes, so that the fetch is issued only locally, from control points immediately above the terminal (scalar) nodes. However, it is a cost of interest for conventional algorithms.

$$\begin{aligned} P_D(0) &= 1; \\ P_D(p+1) &= 1 + P_D(p) = p+2. \\ P_S(p) &= P_D(p) = p+1 = \lg n + 1. \end{aligned}$$

Section 4. Triangular, Clip, and Banded Matrices

A *triangular* matrix either is a non-zero scalar or it decomposes into quadrants, of which the northwest and southeast are triangular, the northeast is zero, and the southwest

is dense (or vice versa, but consistently). The parenthetical postscript here is intended to provide for both "upper" and "lower" triangular, and this recursive definition is to be unfolded consistently in order to preserve one of those 'shapes.' If interpreted freely, it also allows other 'folded' shapes, and the following analysis would still apply to them. Again, Figure 1 illustrates this definition.

A "clip" matrix (Figure 2) looks similar to a triangular matrix, with the role of the main diagonal replaced by one closer to a corner. A measure of this distance is needed, characterized by *bandwidth*, b, where $(b \leq n)$, a measure of the width of the *non*-zero portion of the matrix. In this paper, b will always be 2^w for some integer $w \leq p = \lg n$. When $b = 1$, a clip matrix has a non-zero entry only in the extreme southwest or extreme northeast entry (exclusively). When $b = n$ it is a triangular matrix.

An $n \times n$ *clip* matrix of bandwidth b is either a triangular matrix with $n = b$, or $b < n$ and it has four quadrants, of which three, including the northwest and southeast, are zero and the remaining quadrant is a clip matrix of bandwidth b. Again, the choice of northeast or southwest for the clip is intended to be made consistently over any unfolding of this recursive definition; otherwise, strange foldings arise, but their analyses remain correct.

Intuitively, a banded matrix is zero far away from the main diagonal, but relatively dense in a band of width b from the diagonal. The bandwidth of a matrix, $[a_{i,j}]$, is defined to be

$$\max\{|i-j| \quad | \quad a_{i,j} \neq 0\}.$$

This measure is just under half that of a common alternative that is used to give these forms their Greek names, the full width of the nonzero "band" on either side of—and including—the main diagonal.

An $n \times n$ *banded* matrix of bandwidth b, for $b \leq n$, is either a dense matrix with $b = n$, or has northwest and southeast quadrants being banded matrices, and northeast and southwest quadrants being clip matrices, all of bandwidth b when $b < n$. Once again, it is intended that the clipping be in a consistent pattern similar to Figure 2, but even if this shape is violated, the following analyses still hold. (Dense matrices may be characterized to be banded in two ways: either with $b = n$ or with $b = n - 1$. Since we here restrict n and b to powers of two, however, this confusing redundancy only arises with $n = 2$ and either $b = 2$ or $b = 1$.)

As before, define S_T mapping p to the number of nodes (*space*) necessary to represent a triangular $n \times n$ matrix, and P_T that maps p to the average *path* length in a triangular $n \times n$ matrix. (By convention $p = \lg n$ and $w = \lg b$.) Similarly, S_C maps the pair, $\langle p, w \rangle$, to the space necessary to represent an $n \times n$ clip matrix of bandwidth b, and P_C maps $\langle p, w \rangle$ to the average *path* length in an $n \times n$ clip matrix of bandwidth b. S_B maps the pair, $\langle p, w \rangle$, to the space necessary to represent an $n \times n$ banded matrix of bandwidth b, and P_B maps $\langle p, w \rangle$ to the average *path* length in an $n \times n$ banded matrix of bandwidth b.

For triangular matrices,

$$S_T(p) = S_S(p)$$

$$P_T(0) = 1;$$

$$P_T(p+1) = 1 + \tfrac{1}{4}P_D(p) + \tfrac{2}{4}P_T(p) + \tfrac{1}{4}0.$$

At last, we see the effects of avseraging paths across sparsely represented entries! The path through a non-trivial triangular matrix certainly passes through a root node, and then into one of the four quadrants with equal probability of 25% . Solving [7, Eqn. 2.75],

$$\begin{aligned} P_T(p) &= \sum_{i=0}^{p} 2^{-i} + \frac{p}{4}\sum_{0}^{p-1} 2^{-i} + \frac{1}{4}\sum_{0}^{b-1} i2^{i} \\ &= \tfrac{1}{2}(p + 3 - 2^{-p}) = \tfrac{1}{2}(\lg n + 3 - \tfrac{1}{n}). \end{aligned}$$

As expected, the path length averages to be half the depth of the quadtree, because an isolated half of the tree is **NIL**.

For clip matrices,

$$P_C(p,p) = P_T(p);$$

$$P_C(p+1,w) = 1 + \tfrac{1}{4}P_C(p,w).$$

$$S_C(p,p) = S_T(p);$$

$$S_C(p+1,w) = 1 + S_C(p,w).$$

Solving these, we have

$$\begin{aligned} P_C(p) &= \tfrac{4}{3} + \tfrac{1}{2}\left(\tfrac{2^w}{2^p}\right)^2[\tfrac{1}{3} + w - 2^{-w}] \\ &= \tfrac{4}{3} + \tfrac{1}{2}\left(\tfrac{b}{n}\right)^2[\tfrac{1}{3} + \lg b - \tfrac{1}{b}]. \\ S_C(p) &= p - w + \tfrac{2}{3}(4^w - 1) + 2^w \\ &= \tfrac{2}{3}(b^2 - 1) + b + \lg\left(\tfrac{n}{b}\right). \end{aligned}$$

Banded matrices are composed of clip quadrants and dense quadrants as shown in Figure 2.

$$P_B(p,p) = P_D(p);$$
$$P_B(p+1,w) = 1 + \tfrac{2}{4}P_B(p,w) + \tfrac{2}{4}P_C(p,w).$$
$$S_B(p,p) = S_D(p);$$
$$S_B(p+1,w) = 1 + 2S_B(p,w) + 2S_C(p,w).$$

The general solutions are more complicated:

$$\begin{aligned} P_B(p,w) &= \tfrac{10}{3} + \tfrac{2^w}{2^p}[2(w-1) + (2^{-p} - 2^{-w}) - \tfrac{2^w}{2^p}(w + \tfrac{1}{3})] \\ &= \tfrac{10}{3} + \tfrac{b}{n}[2(\lg b - 1) + (\tfrac{1}{n} - \tfrac{1}{b}) - \tfrac{b}{n}(\lg b + \tfrac{1}{3})]; \\ S_B(p,w) &= \tfrac{4}{3}(2^{p+w+1} + 2^{p-w} - 2^{2w}) + 2[(2^p - 2^w) - (p-w)] - \tfrac{5}{3} \\ &= \tfrac{4}{3}(2nb - b^2 + \tfrac{n}{b} - 1) + 2[n - b - \lg(\tfrac{n}{b})] - \tfrac{1}{3}. \end{aligned}$$

These are most interesting only when we consider particular values of w, to reveal space and average path for specific bandings, *e.g.* tridiagonal matrices:

$$S_B(p,0) = 6n - 2\lg n - 5,$$
$$P_B(p,0) = \frac{10}{3} - \frac{3}{n} + \frac{2}{3n^2};$$

pentadiagonal matrices:

$$S_B(p,1) = 8n - 2\lg n - 9,$$
$$P_B(p,1) = \frac{10}{3} - \frac{1}{n} - \frac{10}{3n^2};$$

and enneadiagonal matrices:

$$S_B(p,2) = 13n - 2\lg n - 26,$$
$$P_B(p,2) = \frac{10}{3} + \frac{7}{n} - \frac{100}{3n^2}.$$

Alternatively, one might define bandwidth, b, of the form $b = 2^v - 1$ for integers v, rather than $b = 2^w$. This allows verification of the above results on tridiagonal matrices ($w = 0; b = 1$), and fills in the following values for heptadiagonal matrices ($b = 3$):

$$\textbf{space} = 13n - 2\lg n - 26,$$
$$\textbf{avgpath} = \frac{10}{3} + \frac{7}{n} - \frac{100}{3n^2}.$$

Thus, a tridiagonal matrix that requires space of at least $3n - 2$ cells in sequential storage requires just under twice that as a quadtree, although that proportion falls as the bandwidth increases. However, expected depth, a reflection of access cost, stays remarkably constant over various bandwidths.

Section 5. Shuffle and FFT Permutation Matrices

Because any permutation matrix can be represented as a vector of integers (regardless of whether a vector is represented using sequential memory or as a binary tree), it seems wasteful even to consider them expanded in a normal matrix representation. They are studied here because they initially seem to be *really* sparse, but turn out to be surprisingly expensive. Permutation matrices, particularly those associated with the fast Fourier transform (FFT) algorithm, have little patterning of zeroes that would allow a collapse of the quadtree representation. Since, as argued elsewhere [3], patterning is necessary to sparseness, it will be interesting to see how these measures of space and path length will compare with those of the highly patterned matrices, already presented.

Because we are more interested in the patterning of zeroes in the permutation matrices than in any space compression possible from sharing within them, all space analyses will ignore the possibility of sharing submatrices. Let a *permutation-patterned matrix* be any $n \times n$ real matrix with n non-zero entries, each row or column of which contains exactly $n - 1$ zero entries. It is then reasonable to presume that each non-zero entry is unique in such a matrix, so that no space savings from sharing of submatrices is possible.

Two permutations, D_p and S_p occur naturally in developing the FFT, which are here called *deal* and *shuffle*, respectively [13]. Multiplying a vector of size $n = 2^p$ by D_p on the left will have the effect of reordering its elements as if the vector were a deck of cards, which was dealt into two full hands of $n/2$ cards which were then stacked. A typical layout of D_p appears below. It is characterized by ones appearing in "knight's moves," first descending from the far northwest entry to central-east, and also ascending from the far southeast entry to central-west.

$$D_p = \left(\begin{array}{ccccccc|ccc} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 1 \end{array}\right)$$

S_p is the inverse of D_p; Pease [6] names S_p as $\mathbf{P}$, without specifying its order. Multiplying a vector on the left by S_p has the effect of performing a perfect riffle-shuffle on the elements of the vector. Multiplying on the right by these permutations reorders the columns similarly.

Figure 2 illustrates how D_p can be restructured into D_{p+1} for $p > 0$. $D_0 = 1$ trivially. by duplicating and restructuring its quadrants. This observation leads to the following two recurrences, expressed using the notation, *SD*, standing for both "shuffle" and "deal," distinguished from "symmetric" or "dense." For the space of a shuffle/deal permutation-patterned matrix:

$$S_{\mathrm{SD}}(0) = 1;$$

$$S_{\mathrm{SD}}(1) = 3;$$

$$S_{\mathrm{SD}}(p+1) = 5 + 2(S_{\mathrm{SD}}(p) - 1).$$

The recurrence for path length is similar. Again, Figure 2 shows how

$$P_{\mathrm{SD}}(0) = 1;$$

$$P_{\mathrm{SD}}(1) = \tfrac{3}{2};$$

$$P_{\mathrm{SD}}(p+1) = 2 + \tfrac{1}{2}(P_S D(p, w) - 1).$$

Solving these recurrences, we find that for $p > 0$

$$S_{\mathrm{SD}}(p) = 3\Big(\sum_{i=0}^{p-3} 2^i\Big) + 2^{p-2} S_{\mathrm{SD}}(2) = 3(2^p - 1) = 3(n-1);$$

$$P_{\mathrm{SD}}(p) = 3(1 - 2^{-p}) = 3(1 - \frac{1}{n}).$$

If sharing were allowed, then $S_{\mathrm{SD}}(p)$ would collapse to $4p - 2$ and $2^i \times 2^i$ shuffle/deal matrices for for *all* integers $i \leq p$ could be represented in shared space $5p - 3$.

The FFT permutation (p-bit-reversal), arises as a consequence of Pease's recurrence [6], given explicitly by Wise [13] who calls it F_p. A key step is to factor out S_p (or D_p) at the left (respectively, right) end at every step in the recurrence; by associating all these factors to the left (right) exclusively of other arithmetic, one of the following two factorizations arises.

$$F_p = S_p \left(\begin{array}{c|c} S_{p-1} & 0 \\ \hline 0 & S_{p-1} \end{array}\right) \left(\begin{array}{c|c} \left(\begin{array}{c|c} S_{p-2} & 0 \\ \hline 0 & S_{p-2} \end{array}\right) & 0 \\ \hline 0 & \left(\begin{array}{c|c} S_{p-2} & 0 \\ \hline 0 & S_{p-2} \end{array}\right) \end{array}\right) \cdots \mathbf{I};$$

$$= \mathbf{I} \cdots \left(\begin{array}{c|c} \left(\begin{array}{c|c} D_{p-2} & 0 \\ \hline 0 & D_{p-2} \end{array}\right) & 0 \\ \hline 0 & \left(\begin{array}{c|c} D_{p-2} & 0 \\ \hline 0 & D_{p-2} \end{array}\right) \end{array}\right) \left(\begin{array}{c|c} D_{p-1} & 0 \\ \hline 0 & D_{p-1} \end{array}\right) D_p.$$

F_p is called "bit-reversal" permutation because it exchanges x_i and $x_{b(i)}$ in permuting $\vec{x}$, where b is a function on natural numbers less than 2^p that reverses the p-bit strings that represent them. That is, the element of $\vec{x}$ indexed $i = \sum_{j=0}^{n} b_j \cdot 2^j$ is indexed $\sum_{j=0}^{n} b_{n-j} \cdot 2^j$ after that permutation. (This fact can be established by a simple induction on p from either factorization above.) Since b is its own inverse, F_p is a symmetric permutation matrix, and so the space measure for F_p that follows might be considerably reduced by allowing for sharing.

If one considers an $n \times n$ FFT permutation-patterned matrix where $n = 4^k$, then the 4^{2k} entries can be perceived as a $2^k \times 2^k$ matrix of blocks, each of which is a $2^k \times 2^k$ matrices. Each of those blocks has *exactly* one element that is non-zero. Thus, its quadtree is gridded as illustrated in Figure 3.

On drawing the block decomposition described above as a tree, we find that the quaternary tree is complete for the first k levels, down to the 4^k *intermediate* notes that are associated with those blocks. The tree rooted at each intermediate node is metalinear; that is, each node has at most one son, along the path toward the unique 1 entry. All other pointers in those subtrees are **NIL**. Figure 3 also illustrates this tree.

Using the notation from the previous sections, let S_F map p onto the *space* required to represent an FFT permutation-patterned matrix, and P_F map p onto the expected *path*

length in an FFT permutation matrix. We have already constrained $n = 4^k$, so $p = 2k$. Then

$$S_F(0) = 1;$$
$$S_F(p+2) = 1 + 4S_F(p) + 2^{p+2}.$$

The last equation arises by adding a root above four FFT permutation-patterned matrices, and extending each metalinear chain by one level at the bottom. Similarly,

$$P_F(0) = 1;$$
$$P_F(p+2) = 1 + P_F(p) + \frac{2^{p+2}}{4^{p+2}}.$$

Solving these two recurrences,

$$S_F(p) = \frac{n \lg n}{2} + \frac{4n}{3} - \frac{1}{3};$$
$$P_F(p+2) = \frac{\lg n}{2} + \frac{4}{3} - \frac{1}{3n}.$$

The results show that the space needed for an $n \times n$ FFT permutation-patterned matrix grows as $\Theta(n \lg n)$, significantly more than the linear space we get with the vector-of-indices representation! The average path length, too, is surprisingly poor, essentially $\frac{\lg n}{2}$, reflecting the completeness of the tree to that level, but already more than half that of a dense matrix!

Theorem 1. *The FFT permutation exhibits the worst case measures of any equivalently-sized permutation-patterned matrix with respect to both space and path length.*

Proof is by an induction like those above. The worst case for space requires that each terminal scalar be isolated in its own subtree, with minimum sharing of its path with any of its cousins; more sharing would reduce total space. Inspection of the tree in Figure 3 shows that the most space is used by completing the shared tree as high in the tree as possible, and hanging n independent paths, each of length, $(\lg n)/2$, beneath . The worst case for average path length occurs when all terminal scalars occur at depth $\lg n$, as they do in the FFT permutation-pattern, even when the tree for $[x\delta_{i,j}]$ can be represented as x. ∎

Section 6. Random Permutations

In this section we find tight bounds for the average time and space required by quadtree representation of random permutation-patterned matrices. Define $P(p)$ to be a $n \times n$ permutation matrix where $n = 2^p$ and p is an integer. As before, we define S_P and P_P that map p to the average number of nodes (space) necessary to represent an $n \times n$ permutation-patterned matrix and the average path length in an $n \times n$ permutation matrix, respectively, where $n = 2^p$, p an integer. We also define $T_D(p)$ to be a complete quadtree obtained for a $2^p \times 2^p$ matrix and $T_P(p)$ to be the quadtree that results from representing a given $2^p \times 2^p$ permutation matrix $P(p)$.

Theorem 2. *If p is even*

$$S_P(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2} + \frac{2^{-p/2}}{1+2^{-p}} + \frac{2 \cdot 2^{-p}}{3 \cdot (1+2^{-p})^2}\right) + 2^p\left(\frac{4}{3} - e^{-\frac{2^{p/2}}{2^{p/2}-2}}\right) - \frac{4}{3}$$

and

$$S_P(p) \geq 2^p\left(\frac{p}{2} + .78\right) - \frac{4}{3}.$$

If p is odd

$$S_P(p) \leq \frac{2^p}{1+2^{-p}}\left(\frac{p}{2} + \frac{1}{2} + \frac{2^{-p/2+1/2}}{1+2^{-p}} + \frac{4 \cdot 2^{-p}}{3 \cdot (1+2^{-p})^2}\right) + 2^p\left(\frac{2}{3} - \frac{1}{2}e^{-\frac{2^{(p+1)/2}}{2^{(p+1)/2}-2}}\right) - \frac{4}{3}$$

and

$$S_P(p) \geq 2^p\left(\frac{p}{2} + .768\right) - \frac{4}{3}.$$

Proof: We only analyze the case p is even. The case p is odd may be established in similar fashion. Associate with each node in $T_D(p)$ a pair of integers (i, j) where i specifies the level of the node in $T_D(p)$ and j is the left to right order of the node at that level. Let $N_P(i, j)$ be an indicator variable which has value 1 if node (i, j) exists in $T_P(p)$ and 0 otherwise. The expectation of $N_P(i, j)$ is the probability that node (i, j) is in $T_P(p)$. Then,

$$S_P(p) = \sum_{i,j} E(N_P(i,j)) = \sum_{i,j} pr(\text{node } (i,j) \text{ is in } T_P(p)). \tag{1}$$

But node (i, j) is in $T_P(p)$ if and only if the submatrix corresponding to that node is not zero. Let $pr(p, i)$ denote the probability that the submatrix corresponding to a node at level i is zero. Therefore, the probability that node (i, j) is in $T_P(p)$ is $1 - pr(p, i)$.

Figure 4 illustrates the calculation of $pr(p,i)$. The $2^{p-i} \times 2^{p-i}$ submatrix corresponding to node (i,j) has been placed at the lower right corner of matrix P for simplicity; this matrix must be entirely zero. In order for P to be a permutation matrix exactly 2^{p-i} rows of submatrix B must contain 1's. This requires 2^{p-i} columns of submatrix A to be all zero. There are $\binom{2^p-2^{p-i}}{2^{p-i}}$ ways to choose 2^{p-i} zero columns in A. For each way to choose 2^{p-i} zero columns in A there are $(2^p - 2^{p-i})!$ ways to place 1's in submatrices A and C such that no two are in the same row or column. Finally, for each way to place 1's in A and C there are $2^{p-i}!$ ways to place 1's in B (these must be placed in the zero columns of A). Thus, for a node at level i,

$$\begin{aligned} pr(p,i) &= \frac{\binom{2^p-2^{p-i}}{2^{p-i}} 2^{p-i}!(2^p - 2^{p-i})!}{2^p!} = \frac{(2^p - 2^{p-i})!(2^p - 2^{p-i})!}{(2^p - 2 \cdot 2^{p-i})!2^p!} \\ &= \frac{(2^p - 2^{p-i} - 0)}{(2^p - 0)} \cdot \frac{(2^p - 2^{p-i} - 1)}{(2^p - 1)} \cdot \frac{(2^p - 2^{p-i} - 2)}{(2^p - 2)} \cdots \frac{(2^p - 2^{p-i+1} + 1)}{(2^p - 2^{p-i} + 1)} \\ &= \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right). \end{aligned} \tag{2}$$

Using (2), the fact that the number of nodes on Level i is 4^i, and that the node on Level 0 is always present, we can rewrite (1) as follows:

$$S_P(p) = 1 + \sum_{i=1}^{p} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right)\right). \tag{3}$$

We now split the sum in (3) into two subsums by cleaving the range of k and derive upper and lower bounds for each. These will later be added to establish the theorem.

a. Consider the sum

$$\begin{aligned} Upper &= \sum_{i=p/2+1}^{p} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right)\right) \\ &\leq \sum_{i=p/2+1}^{p} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1}\right)^{2^{p-i}}\right) \end{aligned}$$

From Lemma 1 of the appendix, $2^{p-i} \geq 1$ implies

$$\left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1}\right)^{2^{p-i}} \geq 1 - \frac{2^{2(p-i)}}{2^p - 2^{p-i} + 1}.$$

Therefore, we can bound *Upper* from above as follows:

$$Upper \leq \sum_{i=p/2+1}^{p} 4^i \left(\frac{4^{p-i}}{2^p - 2^{p-i} + 1} \right) = 2^p \sum_{i=p/2+1}^{p} \frac{1}{1 - 2^{-i} + 2^{-p}}$$

$$= \frac{2^p}{1 + 2^{-p}} \sum_{i=p/2+1}^{p} \frac{1}{1 - \frac{2^{-i}}{1+2^{-p}}}.$$

Making use of the fact that $1/(1-x) \leq 1 + x + 2x^2$ if $0 \leq x \leq 1/2$ we can write

$$Upper \leq \frac{2^p}{1 + 2^{-p}} \sum_{i=p/2+1}^{p} \left(1 + \frac{2^{-i}}{1 + 2^{-p}} + \frac{2 \cdot 2^{-2i}}{(1 + 2^{-p})^2} \right)$$

$$\leq \frac{2^p}{1 + 2^{-p}} \left(\frac{p}{2} + \frac{2^{-p/2}}{1 + 2^{-p}} + \frac{2 \cdot 2^{-p}}{3 \cdot (1 + 2^{-p})^2} \right).$$

Upper is bounded from below by

$$Upper \geq \sum_{i=p/2+1}^{p} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p} \right)^{2^{p-i}} \right).$$

From Lemma 2 of the appendix, $2^{p-i} \geq 1$ implies

$$\left(1 - \frac{2^{p-i}}{2^p} \right)^{2^{p-i}} \leq 1 - \frac{2^{2(p-i)}}{2^p} + \frac{2^{4(p-i)}}{2^{2p+1}}.$$

Therefore,

$$Upper \geq \sum_{i=p/2+1}^{p} 4^i \left(2^{p-2i} - 2^{2p-4i-1} \right) \geq \frac{p2^p}{2} - \frac{2^p}{6}.$$

b. Now consider the sum

$$Lower = \sum_{i=1}^{p/2} 4^i \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k} \right) \right)$$

$$\leq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p - 2^{p-i} + 1} \right)^{2^{p-i}} \right)$$

$$\leq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{1}{2^i - 1} \right)^{2^{p-i}} \right)$$

Lemma 3 of the appendix asserts that $(1-x)^y \geq e^{-xy/(1-x)}$ for all positive x and y.

Therefore,

$$Lower \leq \sum_{i=1}^{p/2} 4^i - \sum_{i=1}^{p/2-1} 4^i \left(1 - \frac{1}{2^i - 1} \right)^{2^{p-i}} - 2^p \left(e^{-\frac{2^{p/2}}{2^{p/2}-2}} \right)$$

$$\leq \frac{2^{p+2}}{3} - \frac{4}{3} - 2^p \left(e^{-\frac{2^{p/2}}{2^{p/2}-2}} \right).$$

Lower is bounded from below by

$$\begin{aligned} \mathit{Lower} &\geq \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{2^{p-i}}{2^p}\right)^{2^{p-i}}\right) \\ &= \sum_{i=1}^{p/2} 4^i \left(1 - \left(1 - \frac{1}{2^i}\right)^{2^{p-i}}\right). \end{aligned}$$

Lemma 4 of the appendix asserts that $(1-x)^{yz} \leq e^{-x^2 y}$ for all positive x and y. So

$$\begin{aligned} \mathit{Lower} &\geq \sum_{i=1}^{p/2} 4^i - 2^p \left(e^{-1} + \frac{1}{4}e^{-4} + \frac{1}{16}e^{-16}\right) - \sum_{i=1}^{p/2-3} 4^i e^{-16} \\ &\geq \frac{2^{p+2}}{3} - \frac{4}{3} - .38 \cdot 2^p. \end{aligned}$$

Putting the bounds of Cases *a* and *b* together proves the theorem. ∎

Corollary 1. *For n a large power of 2, the average space required for $n \times n$ permutation-patterned matrices is between $(n \lg n)/2 + .768n - 4/3$ and $(n \lg n)/2 + .983n - 4/3$.*

Proof: From Theorem 2 the average space gets trapped between the lower bound of $2^p(p/2+.768)-4/3$ and $2^p(p/2+.783)-4/3$ and the upper limit of $2^p(p/2+4/3-e^{-1})-4/3$ and $2^p(p/2+7/6-e^{-1}/2)-4/3$. The result follows. ∎

Theorem 3. *If p is even, then*

$$P_P(p) \leq \frac{p}{2} - e^{-\frac{2^{p/2}}{2^{p/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{1}{3} + \frac{2}{7}\frac{2^{-p}}{1+2^{-p}}\right) + 1$$

and

$$P_P(p) \geq \frac{p}{2} + .862 - \frac{2^{-p}}{3}.$$

If p is odd, then

$$P_P(p) \leq \frac{p}{2} + \frac{1}{2} - e^{-\frac{2^{(p+1)/2}}{2^{(p-1)/2}-2}} + \frac{1}{1+2^{-p}}\left(\frac{2}{3} + \frac{2^{-\frac{p-5}{2}}}{7(1+2^{-p})}\right)$$

and

$$P_P(p) \geq \frac{p}{2} + .764 - \frac{2^{-p}}{3}.$$

Proof: Again we extend only the case where p is even here; the case for odd p is similar. A path contains one node from each of levels 0,1,2... in $T_D(r)$ until a node at some level, say i, representing an all zero submatrix is reached. Therefore,

$$P_P(p) = \sum_{i=0}^{p} pr(\text{random path is at least } i \text{ in length}).$$

But, a path is at least i in length if and only if the level i submatrix is not zero. The probability that the level i submatrix is zero was found in the previous proof. Using that probability we have

$$P_P(p) = 1 + \sum_{i=1}^{p} \left(1 - \prod_{k=0}^{2^{p-i}-1} \left(1 - \frac{2^{p-i}}{2^p - k}\right)\right). \tag{4}$$

As before, we break the sum of (4) into two subsums so that the upper limit of the first subsum is $p/2$ and the lower limit of the second subsum is $p/2+1$ and find bounds for each subsum. Proceeding as in the previous theorem we find that the second subsum is bounded from above by

$$\begin{aligned} Upper &\leq \frac{2^p}{1+2^{-p}} \sum_{i=p/2}^{p} \frac{4^{-i}}{1 - \frac{2^{-i}}{1+2^{-p}}} \\ &\leq \frac{2^p}{1+2^{-p}} \sum_{i=p/2+1}^{p} 4^{-i} \left(1 + \frac{2 \cdot 2^{-i}}{1+2^{-p}}\right) \\ &\leq \frac{2^p}{1+2^{-p}} \left(\frac{1}{3}2^{-p} + \frac{2}{7}\left(\frac{4^{-p}}{1+2^{-p}}\right)\right). \end{aligned}$$

The second subsum is bounded from below by

$$\begin{aligned} Upper &\geq \sum_{i=p/2+1}^{p} \left(2^{p-2i} - 2^{2p-4i}\right) = 2^p \sum_{i=p/2+1}^{p} 4^{-i} - 4^p \sum_{i=p/2+1}^{p} 16^{-i} \\ &\geq \frac{1}{3} - \frac{1}{3 \cdot 2^p} - \frac{1}{15}. \end{aligned}$$

The first subsum is bounded from above:

$$Lower \leq \sum_{i=1}^{p/2} \left(1 - \left(1 - \frac{1}{2^i}\right)^{2^{p-i}}\right) \leq \sum_{i=1}^{p/2} \left(1 - e^{-\frac{2^{p-i}}{2^i - 2}}\right) \leq \frac{p}{2} - e^{-\frac{2^{p/2}}{2^{p/2}-2}}.$$

The first subsum is bounded from below:

$$Lower \geq \sum_{i=1}^{p/2} \left(1 - e^{-2^{p-2i}}\right) \geq \frac{p}{2} - e^{-1} - 2 \cdot e^{-4}.$$

Assembling all the bounds proves the theorem. ∎

Corollary 2. *For n a large power of 2 the average path length in an $n \times n$ permutation matrices is between $\frac{1}{2}\lg(n) + .763$ and $\frac{1}{2}\lg(n) + .966$.*

Proof: From Theorem 3 the average path length tends to between a lower bound of $p/2 + .862$ and $p/2 + .764$ and an upper bound of $p/2 + 4/3 - e^{-1}$ and $p/2 + 7/6 - e^{-1}$. The result follows from $n = 2^p$. ∎

The bounds in both corollaries are tight with respect to their leading coefficients of $\frac{1}{2}$, which coincide with those from the FFT permutation. Their second coefficients lie between 55% and 75% of the corresponding coefficients, both $\frac{4}{3}$, from the FFT permutation. While looser, the two sets of bounds for those second coefficients bracket the same range, which is strikingly centered on the value $\frac{7}{8}$.

Section 7. Measures of Sparsity and Density

Duff states in his authoritative survey, "In quantitative terms, the density of a matrix is defined as the percentage of the number of nonzeros to the total number of entries in the matrix. The term sparsity for the complement of this quantity is rarely used. [3, p. 500]" Rather, he suggests that sparsity of a matrix has as much to do with the distribution of zero elements as with their relative population.

We now have seen closed-form results for total-space and for expected-depth for various patterned and and permutation-patterned matrices. These results are summarized below in Table 1. His perspective is reflected in these results for the space and access-path for familiar kinds of matrices represented as quadtrees.

Based on Duff's *caveat* and these numbers, we present [12, 13] measures of both density and sparsity that are motivated by results on quadtrees, but are defined independently of any particular representation. Their values also appear in Table 1.

Density of a particular matrix is the ratio between the space it occupies, and the space occupied by a dense matrix of the same order. *Non-sparsity* of a particular matrix is the ratio between the expected time to access a random element (path length for quadtrees), and the expected access time within a dense matrix of the same order. *Sparsity* is the difference between one and this non-sparsity measure.

Both density and sparsity are measured on a scale from zero to one. For the conventional row-major, sequential representation of matrices, the density measure corresponds

precisely with Duff's. The sparsity measure is uniformly near zero there, consistent with the observation that this representation offers no special advantage for sparse matrix manipulation.

Let us consider $n \times n$ matrices. Table 1 presents closed-form and asymptotic results for space, density, expected path length (root to terminal node), and sparsity for the classes of matrices treated above. In all cases, a matrix is presumed to be completely dense, except where the indicated pattern requires zeros (or shared storage in the case of symmetry).

	Space	Density*	Expected Path	Sparsity*
Dense	$\frac{4}{3}(n^2 - \frac{1}{4})$	1	$\lg n + 1$	0
Symmetric	$\frac{2}{3}(n+2)(n-\frac{1}{2})$	$\frac{1}{2}$	$\lg n + 1$	0
Triangular	$\frac{2}{3}(n+2)(n-\frac{1}{2})$	$\frac{1}{2}$	$\frac{\lg n}{2} + \frac{3}{2} - \frac{1}{2n}$	$\frac{1}{2} - \frac{1}{\lg n}$
FFT permutation	$\frac{n \lg n}{2} + \frac{4n}{3} - \frac{1}{3}$	$\frac{3}{8}\frac{\lg n}{n}$	$\frac{\lg n}{2} + \frac{4}{3} - \frac{1}{3n}$	$\frac{1}{2} - \frac{0.83}{\lg n}$
Random permutation	$\frac{n \lg n}{2} + 0.87n - \frac{4}{3}$	$\frac{3}{8}\frac{\lg n}{n}$	$\frac{\lg n}{2} + 0.87$	$\frac{1}{2} - \frac{0.37}{\lg n}$
Tridiagonal	$6n - 2\lg n - 5$	0	$\frac{10}{3} - \frac{3}{n} + \frac{2}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Pentadiagonal	$8n - 2\lg n - 9$	0	$\frac{10}{3} - \frac{1}{n} - \frac{10}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Heptadiagonal	$11n - 2\lg n - 19$	0	$\frac{10}{3} + \frac{5}{n} - \frac{76}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Enneadiagonal	$13n - 2\lg n - 26$	0	$\frac{10}{3} + \frac{7}{n} - \frac{100}{3n^2}$	$1 - \frac{3.33}{\lg n}$
Shuffle permutation	$3(n-1)$	0	$3(1 - \frac{1}{n})$	$1 - \frac{3}{\lg n}$
Identity	1	0	1	$1 - \frac{1}{\lg n}$

Table 1. Measures of patterned and unpatterned matrices as quadtrees.

**Density* is accurate within a term of $\Theta(n^{-1})$. *Sparsity* is accurate within a term of $\Theta((\lg n)^{-2})$.

The remarkable entry in the table is for random and FFT permutations-patterned, which measure out to be neither dense nor sparse, in spite of the fact that they only contains n non-zeroes of n^2 entries. This is consistent with Duff's observation that patterning is essential to sparseness; the bit-reversal permutation is characterized by its lack of local patterning! It is fortunate that we already prefer an alternative representation for permutation matrices that is not dense: as vectors of integers.

Section 8. Analysis of Matrix Addition

The utility of any measure must be demonstrated in analytical or in experimental studies of the behavior of algorithms on arguments with various measures. This section offers an example worst-case analysis of the matrix addition, the simplest matrix algirithm. The measure of sparsity has not been well studied, however, and more work should be done.

Theorem 4. *The sum of two $n \times n$ addends, respectively of density d_1 and d_2 and of sparsity s_1 and s_2, requires uniprocessor (proportional) time and space within the bounds:*

$$n^2(\lg n + 1)\max[0, 1 - (s_1 + s_2)] = \textbf{uniprocessor time} = n^2(\lg n + 1)[1 - \max(s_1, s_2))],$$

$$\tfrac{4}{3}n^2|d_1 - d_2| \leq \textbf{space}_{\textbf{sum}} < \tfrac{4}{3}n^2 \min(1, d_1 + d_2);$$

and, itself has sparsity and density measures within the bounds,

$$\max(0, s_1 + s_2 - 1) \leq \textbf{sparsity}_{\textbf{sum}} \leq 1 - |s_1 - s_2|;$$

$$|d_1 - d_2| \leq \textbf{density}_{\textbf{sum}} < \min(1, d_1 + d_2).$$

Proof: These results follow from the following observations. The sum will be, at worst, dense:

$$\textbf{space}_{\textbf{sum}} \leq S_D(\lg n) \leq \tfrac{4}{3}n^2 - 1/3 < \tfrac{4}{3}n^2,$$

and is no larger than the sum of the space occupied by the addends: shape

$$\textbf{space}_{\textbf{sum}} \leq \textbf{space}_1 + \textbf{space}_2 \leq S_D(\lg n)(d_1 + d_2).$$

If one addend is the negative of the other, then the space for the sum, **NIL**, is zero;

$$\textbf{space}_{\textbf{sum}} \geq |space_1 - space_2| = 0 = \tfrac{4}{3}n^2|d_1 - d_2|;$$

otherwise the sum-tree must, at least contain a root:

$$\begin{aligned}\textbf{space}_{\textbf{sum}} &\geq |space_1 - space_2| + 1 \\ &\geq \tfrac{4}{3}(n^2 - 1/4)|d_1 - d_2| + 1 > \tfrac{4}{3}n^2|d_1 - d_2|\end{aligned}$$

Let (proportional) time be measured by the number of nodes visited in computing the sum quadtree; only the bases of the trees, common to both addends need be traversed becaused unshared periphery can be borrowed in the sum. The number of nodes visited

by an addition has is zero, at best, and and is at least proportional to the path common to the two addends:

$$\begin{aligned}\textbf{uniprocessortime} &\geq 0;\\ &\geq \textbf{totalpath}_1 + \textbf{totalpath}_2 - \textbf{totalpath}_D;\\ &\geq n^2(\lg n + 1)\max(0, 1 - (s_1 + s_2)).\end{aligned}$$

It is less than the lesser of the two total paths:

$$\begin{aligned}\textbf{uniprocessortime} &\leq \min(\textbf{totalpath}_1, \textbf{totalpath}_2)\\ &\leq n^2(\lg n + 1)(1 - \max(s_1, s_2)).\end{aligned}$$

The density bounds follow from dividing the bounds from the cases considered in the space analysis, above, by $S_D(\lg n)$. In considering sparsity, we must consider the entire, unshared path length in the sum:

$$\begin{aligned}\textbf{totalpath}_{\textbf{sum}} &\geq 0;\\ &\leq n^2(\lg n + 1);\\ &\geq |\textbf{totalpath}_1 - \textbf{totalpath}_2|;\\ &\leq \textbf{totalpath}_1 + \textbf{totalpath}_2.\end{aligned}$$

the first two bounds occur when the sum is *NIL* or completely dense, respectively. The second pair of bounds account for the extreme cases where the addends are nearly additive inverses, or have no coincident non-zero elements. Since, for $i = 1, 2$:

$$\textbf{totalpath}_i = n^2(\lg n + 1)(1 - s_i),$$

$$\max(0, s_1 + s_2 - 1) = 1 - \min(1, 2 - (s_1 + s_2)) \leq \textbf{sparsity} = 1 - \frac{\textbf{totalpath}}{n^2(\lg n + 1)} \leq 1 - |s_1 - s_2|.$$

In general, however, analytical results like these are difficult and so the utility of these measures ultimately must be established or denied by experimentation on real data.

Section 9. Conclusions

Measures of space and expected depth of quadtree representations of various kinds of "sparse" matrices have been presented. All measures exceed those for conventional representations, in most cases by only a linear factor. Quadtree representation, however,

offers a facility for process decomposition and scheduling that is unavailable with other representations, and so the increased costs may easily be recoverable.

These measures have been developed analytically from familiar cases. Their utility depends upon two kinds of experimental investigation on genuine data. The two questions to be addressed are whether these measures separate among cases that actually arise in real data, and whether the performance of real algorithms can be reflected in these measures.

Along with experimental identification of populations of likely patterns, we should seek analytic results, particularly of average cases because the sparsity measure is based on an expected value. Results on multiplication and matrix inversion or solving linear systems, similar to those given here in Section 8, would be quite useful.

While the measures apply as well to data stored sequentially on uniprocessors, it is particularly important to apply these measures to multiprocessor algorithms because multiprocessing motivates the quadtree representation that indirectly motivates them. Only with many processors can the overhead inherent in this representation be recovered.

One might speculate that the widespread use of permutations can cloud these experiments. It is not appropriate to study data after it has been artificially permuted to suit a particular uniprocessor algorithm or architecture. While it is usual to permute matrices on uniprocessors in order to bring the data into a desirable pattern, there will be reason to avoid repeated permutations in a rich multiprocessing environment. There is a very simple path between memory and processor on a uniprocessor, so it is difficult to foresee addressing patterns hampering its effective bandwidth.

On multiprocessors, however, one can anticipate that the pattern of many processors contending for access to many memories can reduce that effective bandwidth in extreme cases. The simplest parallel algorithm that can raise such contention is an unpatterned, random, permutation of a vector stored across several memory banks. It is difficult to believe that each bank will be regularly accessed without delay by other processors accessing the same portion of the address space. Whether the poor values for permutation matrices is a reflection of this problem remains to be proved.

If these measures are a harbinger of that difficulty, however, they also point to a likely alternative to heavy use of permutations. Perhaps, it will be desirable to avoid wild permutations (like the FFT) on parallel processors, in favor of alternatives like factoring

simpler permutations from partial results. Although the FFT permutation-pattern measured poorly, it admits a factorization of Shuffles (or Deals) each of which measures out to be comparatively tame. Perhaps these permutations can be distributed out from a parallel process to a serial process (like input/output, usually on uniprocessors) where they would not create bandwidth-consuming, chaotic access. Perhaps, like the permutations within the fast Fourier transform applied to convolution problems, these lazy permutations might simplify or cancel themselves by subsequent associativity with other postponed permutations.

Acknowledgement: The final draft of this paper was assembled using facilities at the Computer Science Lab of Tektronix, Inc. Section 6 and the Appendix belong to the second author.

References

1. S. K. Abdali. & D. D. Saunders. Transitive closure and related semiring properties via eliminants. *Theoretical Computer Science* **40**, 2,3 (1985), 257-274.

2. P. J. Denning Parallel computing and its evolution. *Comm. ACM* **29**, 12 (December, 1986), 1163-1167.

3. I. S. Duff. A survey of sparse matrix research. *Proc. IEEE* **65**, 4 (April, 1977), 500-535.

4. S. D. Johnson. "Storage Allocation for List Multiprocessing", Indiana University Computer Science Dept. Technical Report No. 168, (March, 1985).

5. D. E. Knuth. *The Art of Computer Programming,* **I**, *Fundamental Algorithms*, 2nd Ed., Reading, MA, Addison-Wesley (1975), 299-318 + 401, 556.

6. M. C. Pease. An adaptation of the fast Fourier transform for parallel processing. *J. ACM* **15**, 2 (April, 1968), 252-264.

7. P. W. Purdom, Jr., & C. A. Brown. *The Analysis of Algorithms*, New York, Holt, Rinehart and Winston (1985).

8. R. Rettberg & R. Thomas. Contention is no obstacle to shared-memory multiprocessing. *Comm. ACM* **29**, 12 (December, 1986), 1202-1212.

9. J. R. Rice. *Matrix Computations and Mathematical Software,* New York, McGraw-Hill (1981), Chapter 4.

10. V. Strassen. Gaussian elimination is not optimal. *Numer. Math.* **13**, 4 (August, 1969), 354-356.

11. D. S. Wise. Representing matrices as quadtrees for parallel processors. *Information Processing Letters* **20** (May, 1985), 195-199.

12. D. S. Wise. Parallel decomposition of matrix inversion using quadtrees. *Proc. 1986 International Conference on Parallel Processing* (IEEE Cat. No. 86CH2355-6), 92-99.

13. D. S. Wise. Matrix algebra and applicative programming. **3rd Intl. Conf. on Functional Programming Languages and Computer Architecture**, Berlin, Springer (September, 1987), in press.

14. M. F. Young. A functional language and modular arithmetic for scientific computing. In Jean-Pierre Jouannaud (ed.), *Functional Programming Languages and Computer Architecture, Lecture Notes in Computer Science* **201**, Berlin, Springer (1985), 305-318.

Appendix

Lemma 1. *If $b \geq 1$ and $|a| \leq 1$ then $(1-a)^b \geq 1 - ab$.*

Proof: Compare the Taylor Series expansion of the logarithm of both sides.∎

Lemma 2. *If $a \geq 1$ then $(1-x)^a \leq 1 - ax + (ax)^2/2$.*

Proof: Using Taylor's Series expansion (around zero) with remainder [7, p. 150] we can write

$$(1-x)^a = 1 - ax + \frac{a^2(1-c)^a x^2}{2}$$

where $0 \leq c \leq x$. Setting $c = 0$ proves the lemma. ∎

Lemma 3. *For all $x, y \geq 0$, $(1-x)^y \geq e^{-xy/(1-x)}$.*

Proof: It suffices to show that $-\ln(1-x) \leq x/(1-x)$. But

$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} \ldots$$

and

$$\frac{x}{1-x} = x + x^2 + x^3 + x^4 + \ldots$$

Straightforward comparison proves the lemma. ∎

Lemma 4. *For all $x, y \geq 0$, $(1-x)^{xy} \leq e^{-x^2 y}$.*

Proof: Take the logarithm of both sides, apply the Taylor Series expansion to the left side and compare terms. ∎

END

11-87

DTIC